Software Engineering I (02161)

Week 9: Layered Architecture Persistency Layer; Software Development Process

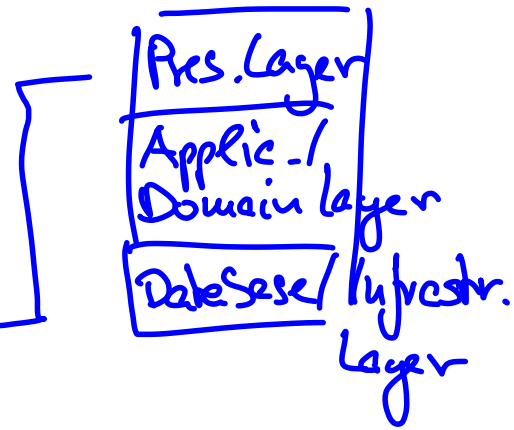
Assoc. Prof. Hubert Baumeister

DTU Compute
Technical University of Denmark

Spring 2017

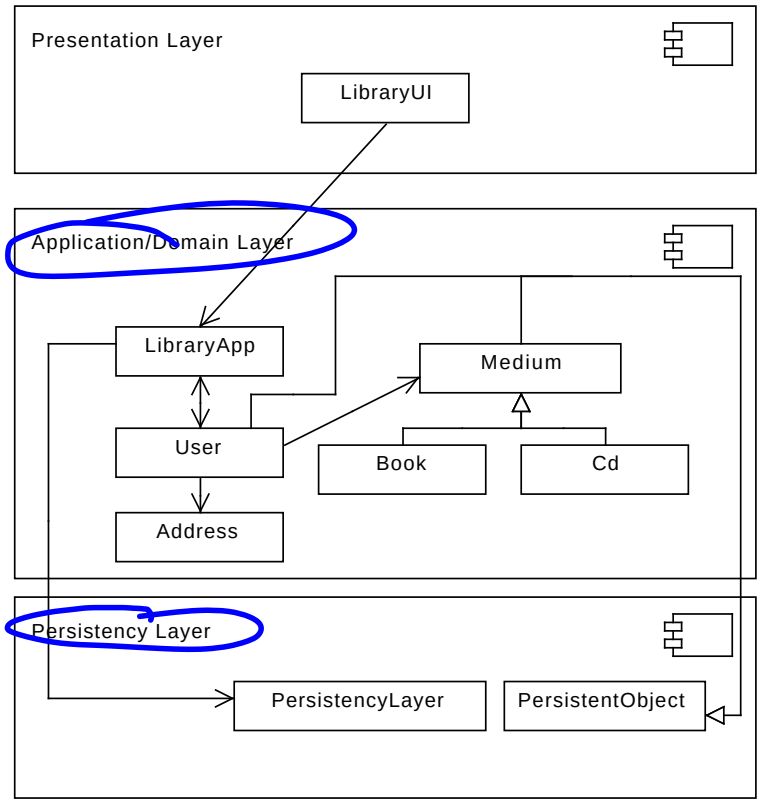
Recap

- Previously
  - Design: From requirements to design, class diagrams, sequence diagrams, low coupling and high cohesion: layered architecture basics
- Last Week
  - Version Control
  - State machines: UML Behaviour diagram
  - Layered Architecture: Presentation Layer
- This week
  - Layered Architecture: Persistency Layer
  - Software Development Processes / Project Planning



Contents

Layered Architecture: Persistency Layer for the library application



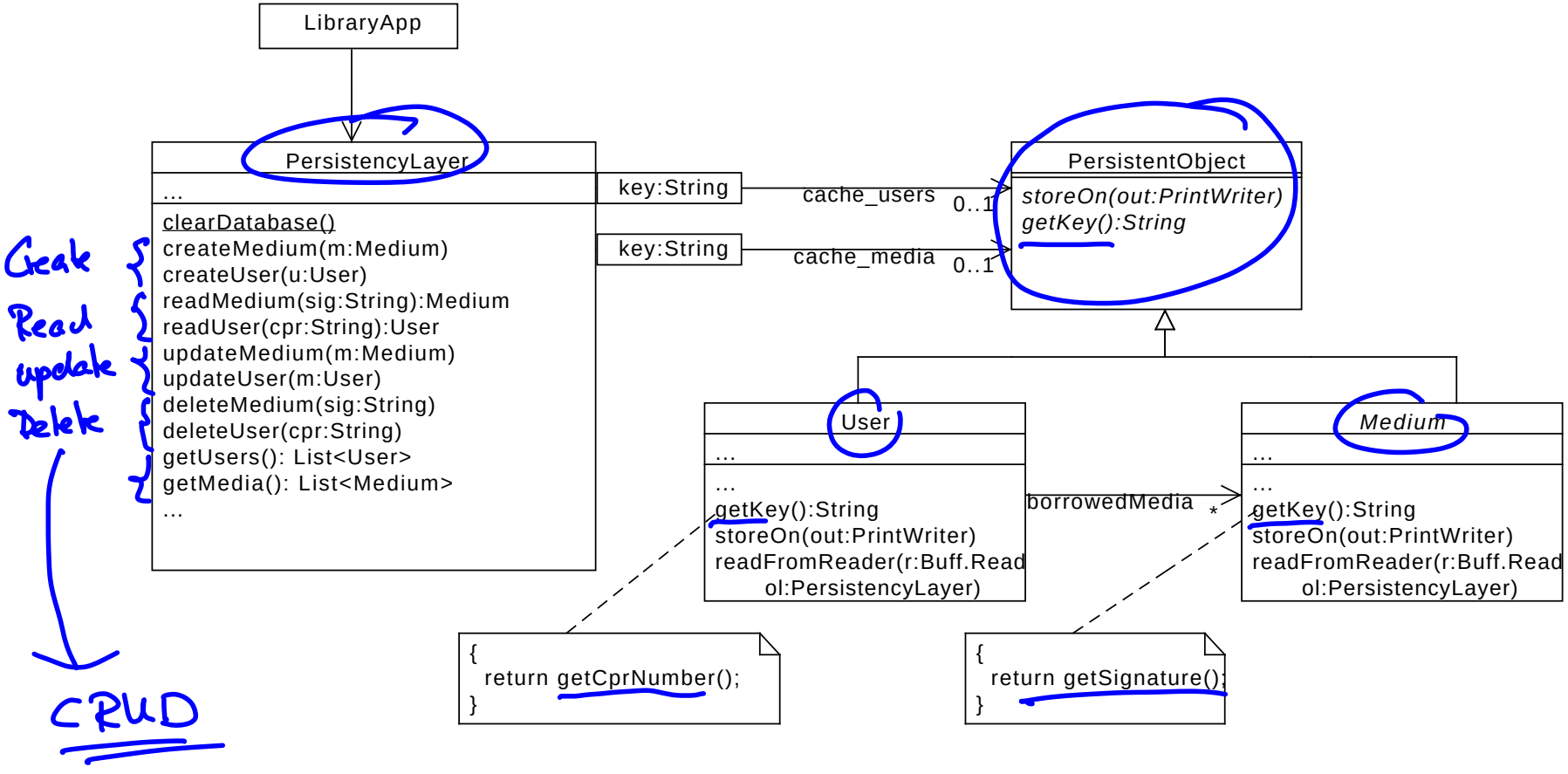
- Data stored in two files `users.txt` & `media.txt`; address has no file
- A book

```
dtu.library.app.Book
b01
some book author
some book title
Mar 13, 2011
<empty line>
```

- A user

```
dtu.library.app.User
cpr-number
Some Name
a@b.dk
Kongevejen
2120
Hellerup
b01
c01
<empty line>
```

Persistency Layer



Layered Architecture: Persistency Layer for the library application

PersistencyLayer
cache_users cache_medium
clearDatabase() createMedium(m:Medium) createUser(u:User) readMedium(sig:String):Medium readUser(cpr:String):User updateMedium(m:Medium) updateUser(m:User) deleteMedium(sig:String) deleteUser(cpr:String) getUsers(): List<User> getMedia(): List<Medium> ...

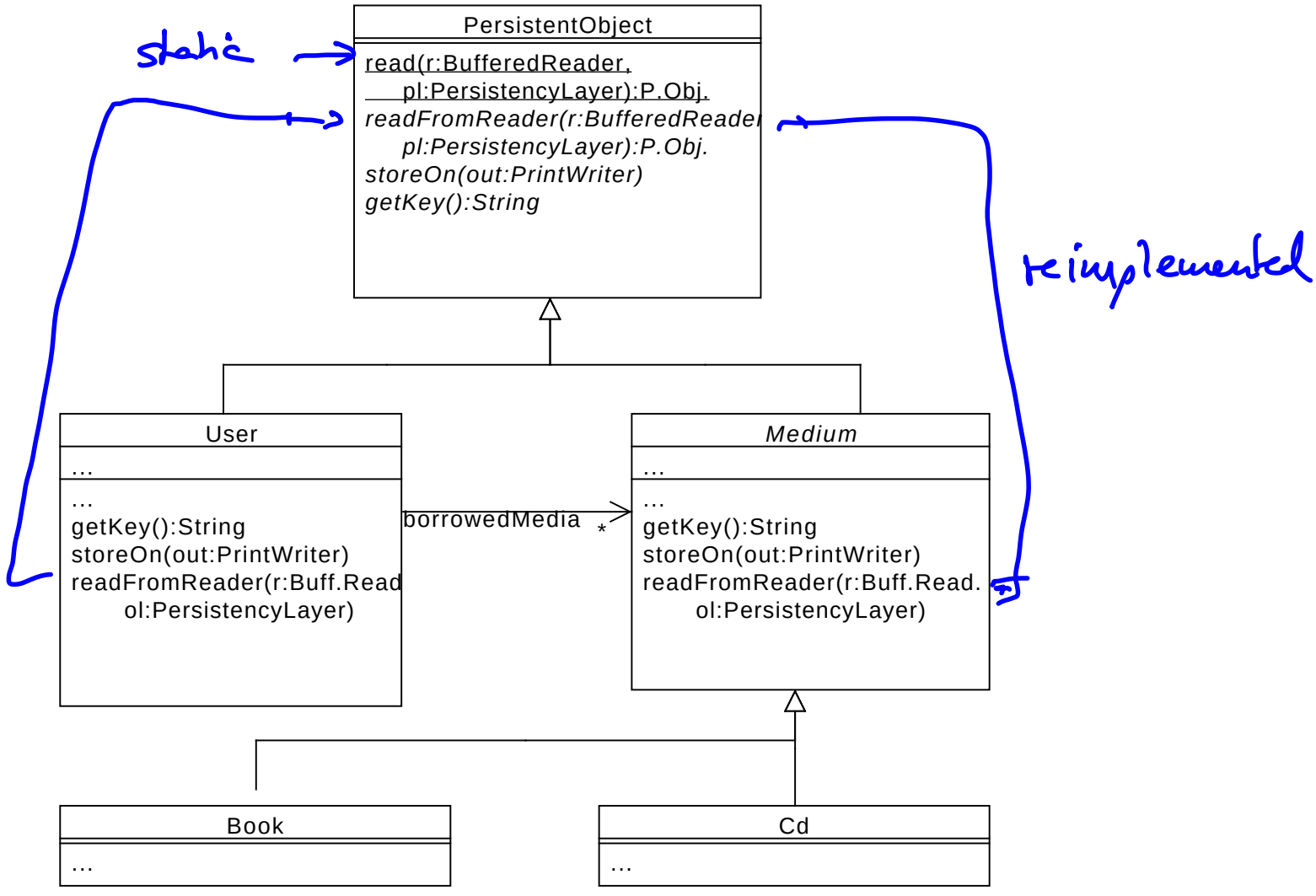
- clearDatabase:removes the text files to create an empty database. Used with tests in @Before methods: Independent tests
- createMedium/User: appends a new record to the corresponding file
- updateMedium/User: copy all entries in a new file; replace the old entry with the new entry on copying
- deleteMedium/User: do the same as updateMedium/User, but don't copy the object to be deleted

→ What is the complexity of
createMedium/User, $O(1)$
readMedium/User, $k = \frac{1}{2}n$ in average
updateMedium/User, $O(n) \rightsquigarrow kn$
deleteMedium/User? $O(n)$

n = number of object

$O(n)$

Reading/Writing User and Media objects



Reading User and Media objects

```
public class PersistentObject {
  ...
   public static PersistentObject read(BufferedReader in,
       PersistencyLayer pl) throws IOException {
     String type = in.readLine();
     PersistentObject po = null;
     po = (PersistentObject) Class.forName(type).newInstance();
     return po.readFromRader(pl, reader);
   }
  ...
}
```

- Delegate the initialization of the created object to the newly created object
  - → No access to the instance variables of the object
  - → The object knows best which data it needs

```
PersistentObject po = new User();
po.readFromReader(pl,in);
```

- instead of

```
PersistentObject po = new User();
po.cprNumber(in.readLine());
po.name(in.readLine());
 ...
```

Class User

```
public class User {
  ...
  public void readFromReader(PersistencyLayer pl, BufferedReader in)
      throws IOException {
    cprNumber = in.readLine(); name = in.readLine();
    email = in.readLine();
    address = Address.readFrom(in);
    borrowedMedia = new ArrayList<Medium>();
    String signature = in.readLine();
    while (!signature.isEmpty()) {
      borrowedMedia.add(pl.readMedium(signature));
      signature = in.readLine();
    }
  }
}
```

empty line terminates

```
dtu.library.app.User
cpr-number
Some Name
a@b.dk
Kongevejen
2120
Hellerup
b01
c01
<empty line>
```

Use of Files

Writing files

```
FileWriter fw = new FileWriter(filename, true);
                      // true = append; false = replace
PrintWriter out = new PrintWriter(fw);
out.println("Some line");
out.print("Some string without new line");
```

Reading files

```
FileReader fr = new FileReader(filename);
BufferedReader in = new BufferedReader(fr);
String line - in.readLine();
```

Deleting and renaming files

```
File f = new File(filename);
f.delete();
f.renameTo(new File(new_filename));
```

Tests for the integration

```
@Before
public void setUp() throws Exception {
  libApp = new LibraryApp();
  PersistencyLayer.clearDatabase();
  libApp.adminLogin("adminadmin");
  Address address = new Address("Kongevejen", 2120, "Hellerupl");
  user = new User("cpr-number", "Some Name", "a@b.dk", address);
  libApp.register(user);
  b = new Book("b01", "some book title", "some book author");
  c = new Cd("c01", "some cd title", "some cd author");
  libApp.addMedium(b);
  libApp.addMedium(c);
}
```

test-data

```
@Test
public void testBorrowing() throws Exception {
  user.borrowMedium(b);
  user.borrowMedium(c);
  PersistencyLayer pl = new PersistencyLayer();
  User user1 = pl.readUser(user.getCprNumber());
  assertEquals(2, user1.getBorrowedMedia().size());
  Utilities.compareUsers(user, user1);
}
```

DataSave test

Implementation in LibraryApp

```
public void borrowMedium(Medium medium) throws BorrowException {
  if (medium == null)
    return;
  if (borrowedMedia.size() >= 10) {
    throw new TooManyBooksException();
  }
  for (Medium mdm : borrowedMedia) {
    if (mdm.isOverdue()) {
      throw new HasOverdueMedia();
    }
  }
  medium.setBorrowDate(libApp.getDate());
  borrowedMedia.add(medium);
  try {
    libApp.getPersistencyLayer().updateUser(this);
  } catch (IOException e) {
    throw new Error(e);
  }
}
```

Application logic

Database update

IOException

Issues: Object identity

user1.setName("...")
pl.update(user2);

```
PersistencyLayer pl = new PersistencyLayer();
User user1 = pl.readUser("12345");
User user2 = pl.readUser("12345");
```

Which assertion is true?

```
assertNotSame(user1,user2)
assertSame(user1,user2)
```

✓

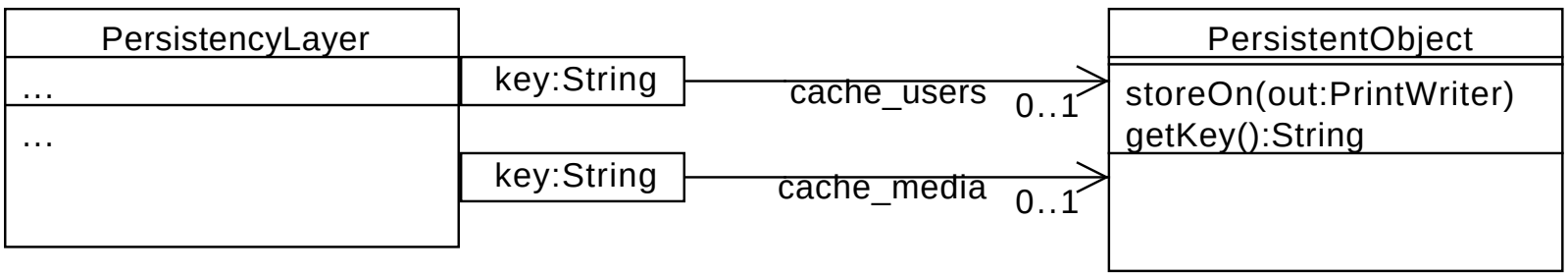
happens in a naïve implementation

we could like this

Solution: Qualified Associations / Maps

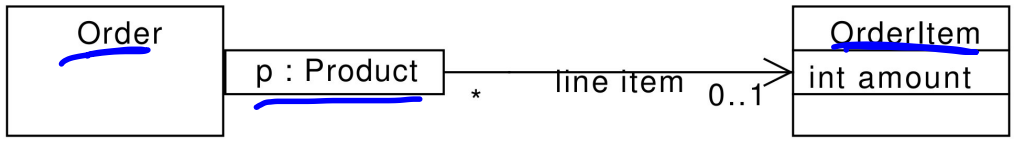
```
Map<String,PersitentObject> cacheUsers =
                            new HashMap()<String,PersistentObject>
Map<String,PersitentObject> cacheMedia =
                            new HashMap()<String,PersistentObject>
```

UML Notation




```
public User readUser(String key) {
   if (cacheUsers.contains(key)) { return cacheUsers.get(key); }
   User user = readUserFromFile(String key);
   if (user != null) { cacheUsers.put(key,user); }
   return user;
}
```

Qualified Assocations I

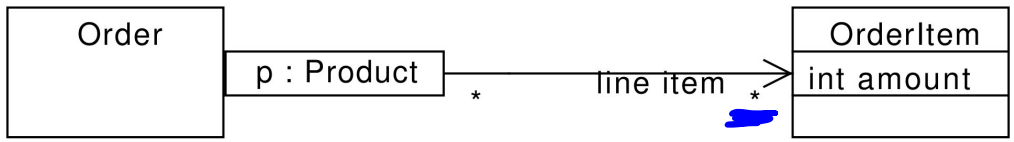


- A qualified association is an association, where an object is associated to another object via a qualifier (a third object)
- An **Order** has an **OrderItem** for each **product**
- If the multiplicity is ≤ 1 then an order has at most one list item for each product
  - $\rightarrow$ This is usually implemented by a **map** or **dictionary** mapping products to order items

```
public class Order {
   private Map<Product,OrderItem>
      listItem = new HashMap<Product,OrderItem>()
   ...
}
```

Qualified Associations II

- If the multiplicity is *, then several order items may be associated to a product



- Then the map has to return a collection for each product

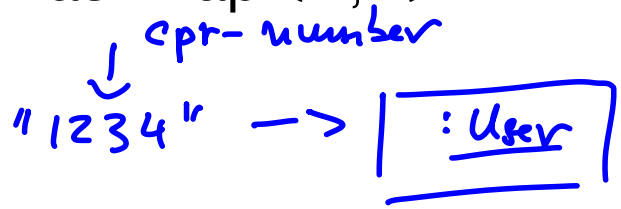
```
public class Order {
   private Map<Product,Collection<OrderItem>>
      listItems = new HashMap<Product,Collection<OrderItem>>()
   ...
}
```

Map<K,V> Interface

- Dictionary (table): keys of type *K*, values of type *V*
- Implementation class: HashMap<K,V>
- Operations
  - `m.put(aK,aV)`
  - `m.get(aK)`
  - `m.containsKey(aK)`
- Properties
  - *aK* is not a key in *m*

    ```
    assertFalse(m.containsKey(aK));
    assertNull(m.get(aK));
    ```
  - Value *aV* is added with key *aK* to *m*

    ```
    m.put(aK,aV);
    assertTrue(m.containsKey(aK));
    assertSame(aV,m.get(aK));
    ```



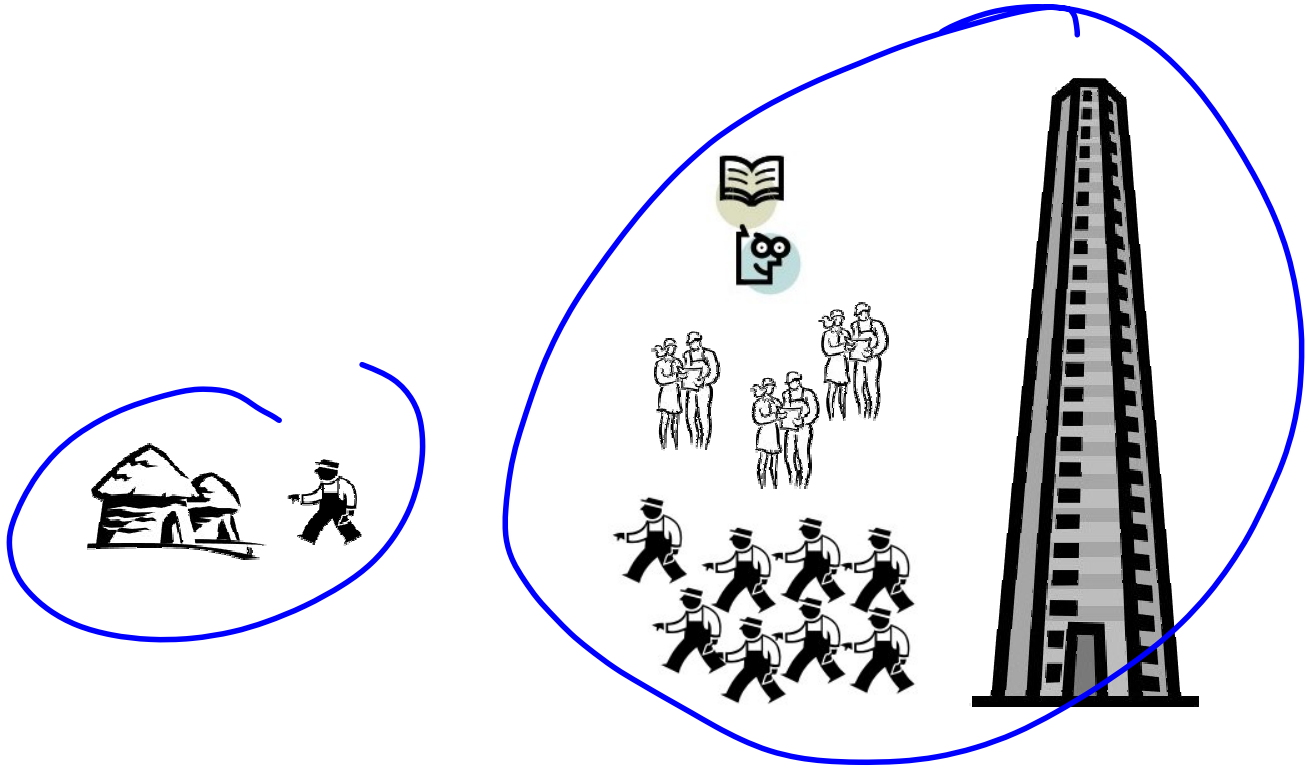
Programming exercise 6:

1) Implement the persistency layer (tests provided)
2) Intergrate persistentcy layer in the library application (tests have to be written)

- Additional information
  `http://www2.imm.dtu.dk/courses/02161/2017/slides/pe_persistency.pdf`
- Solution: `http://www2.imm.dtu.dk/courses/02161/2017/files/library07_solution.zip`

Contents

Software Development Challenges



- Challenges of Software Development
  - On **time**
  - In **budget**
  - No **defects**
  - Customer **satisfaction**

Software Development Process

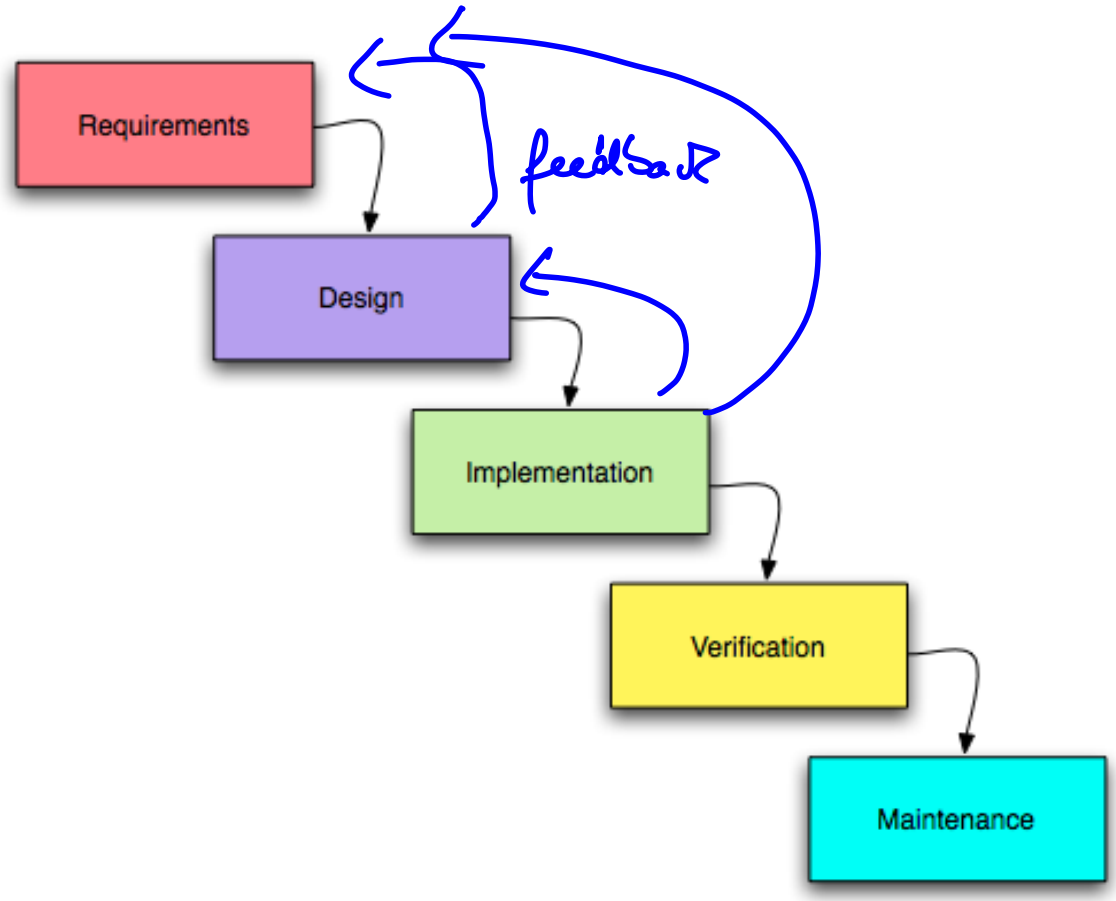
- Activities in Software Development
  - *Understand* and *document* what the customer wants: **Requirements Engineering**
  - *How* to build the software: **Design**
  - *Build* the software: **Implementation**
  - *Validate*: **Testing**, **Verification**, **Evaluation**

→ Set of techniques: Use cases, CRC cards, refactoring, test-driven development, ...

- How to apply the techniques:

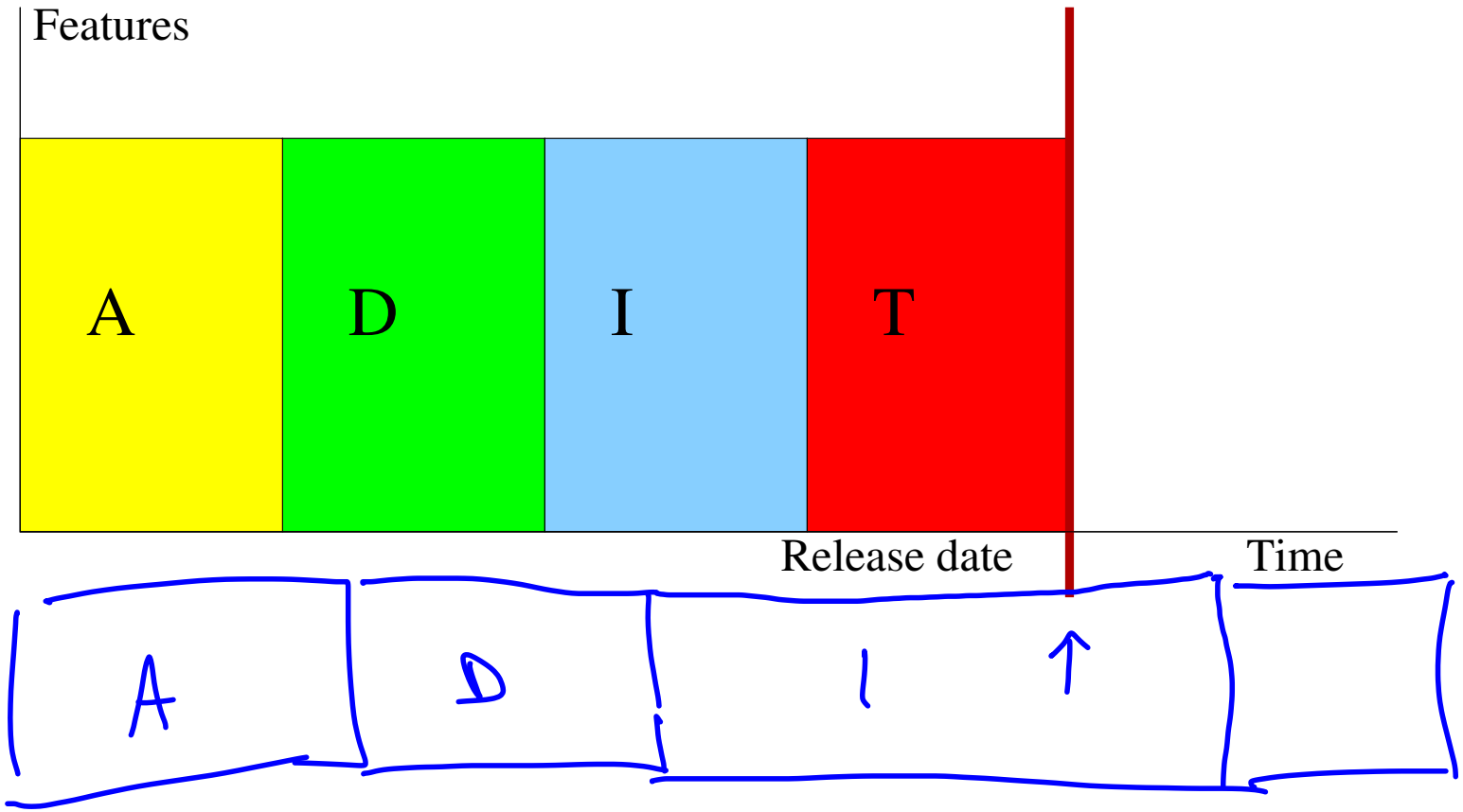
→ Different **software development processes**: Waterfall, Iterative processes, agile, lean, ...

Waterfall process

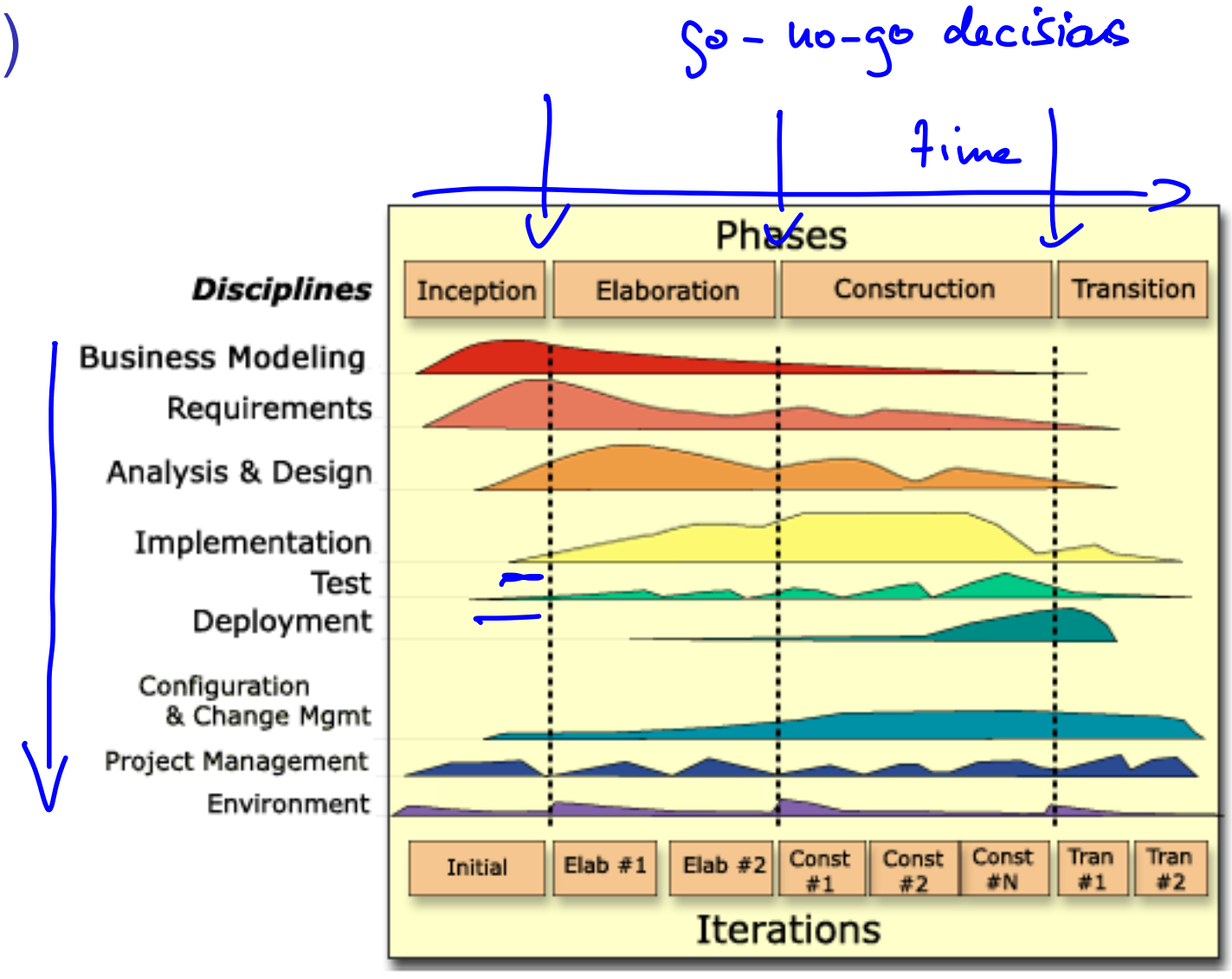


- 1970: Used by Winston W. Royce in a article as a an example of how not to develop software
- 1985: Waterfall was required by the United States Department of Defence from its contractors

Delays in waterfall processes

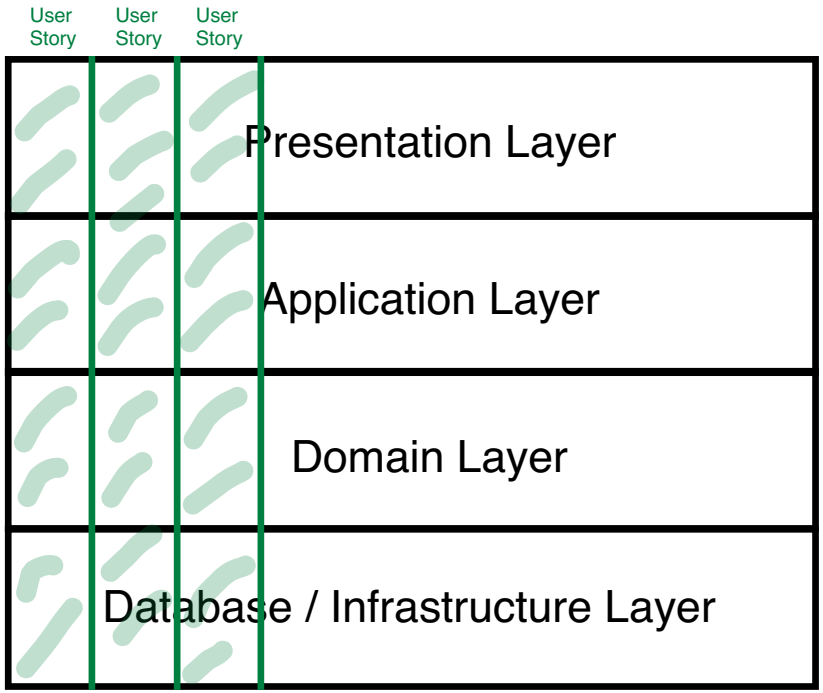


Iterative Processes: E.g. (Rational) Unified Process (1996)



Agile Software Development Methods (1999)

- Extreme Programming (XP) (1999), Scrum (1995–2001), Feature Driven Development (FDD) (1999), Lean Software Development (2003), . . .
- Kanban (2010): often seen as a method, but it is a tool to improve processes and is to be used with the other processes
- Based on the Agile Manifesto (2001)



- Highest priority user story first
- If delayed: important features are implemented

Manifesto for Agile Software Development (2001)

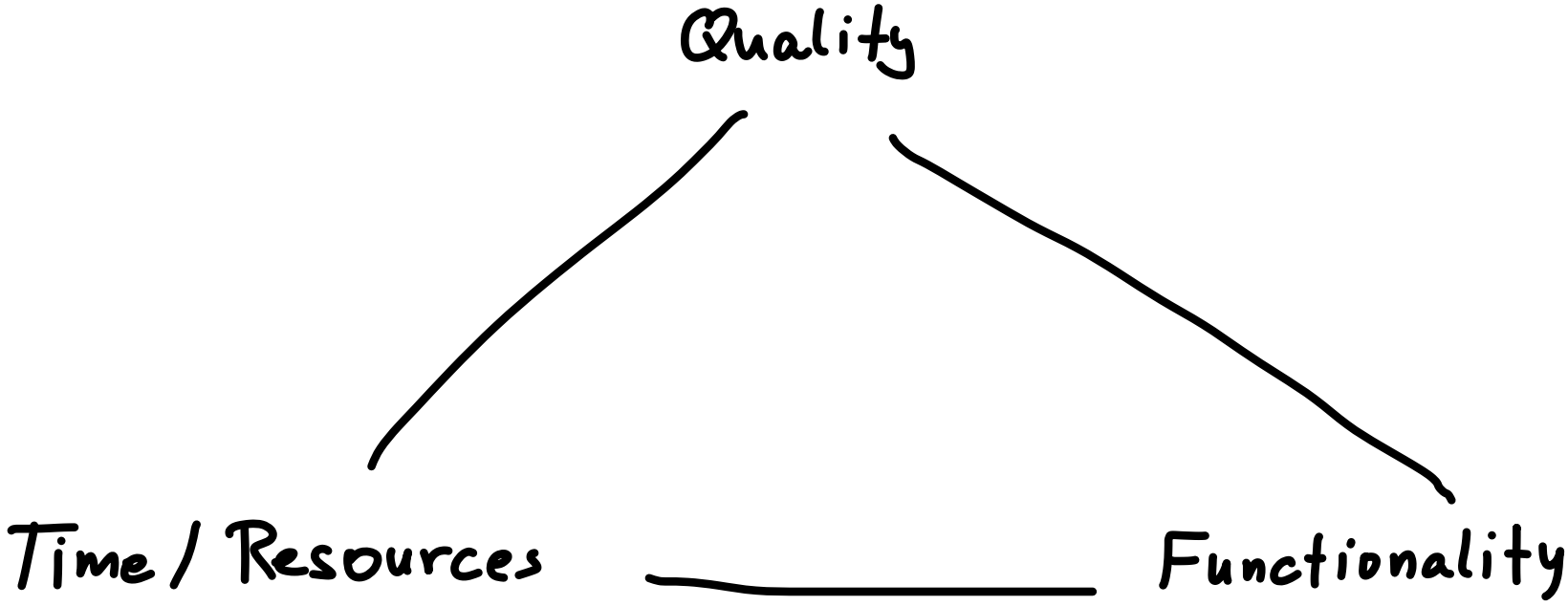
"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- *Individuals and interactions over processes and tools*
- *Working software over comprehensive documentation*
- *Customer collaboration over contract negotiation*
- *Responding to change over following a plan*

That is, while there is value in the items on the right, we value the items on the left more."

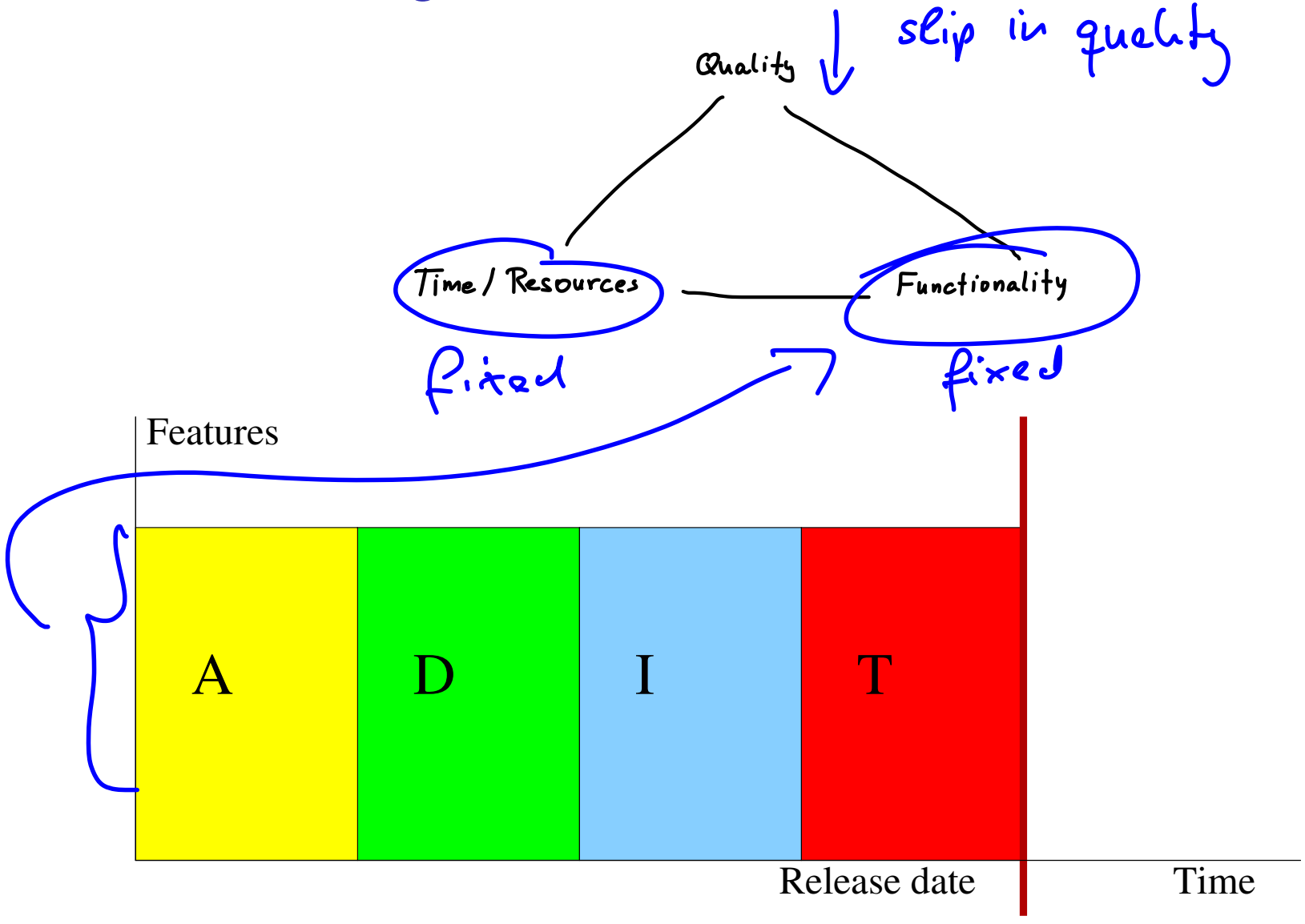
`http://www.agilemanifesto.org`

Resource Triangle

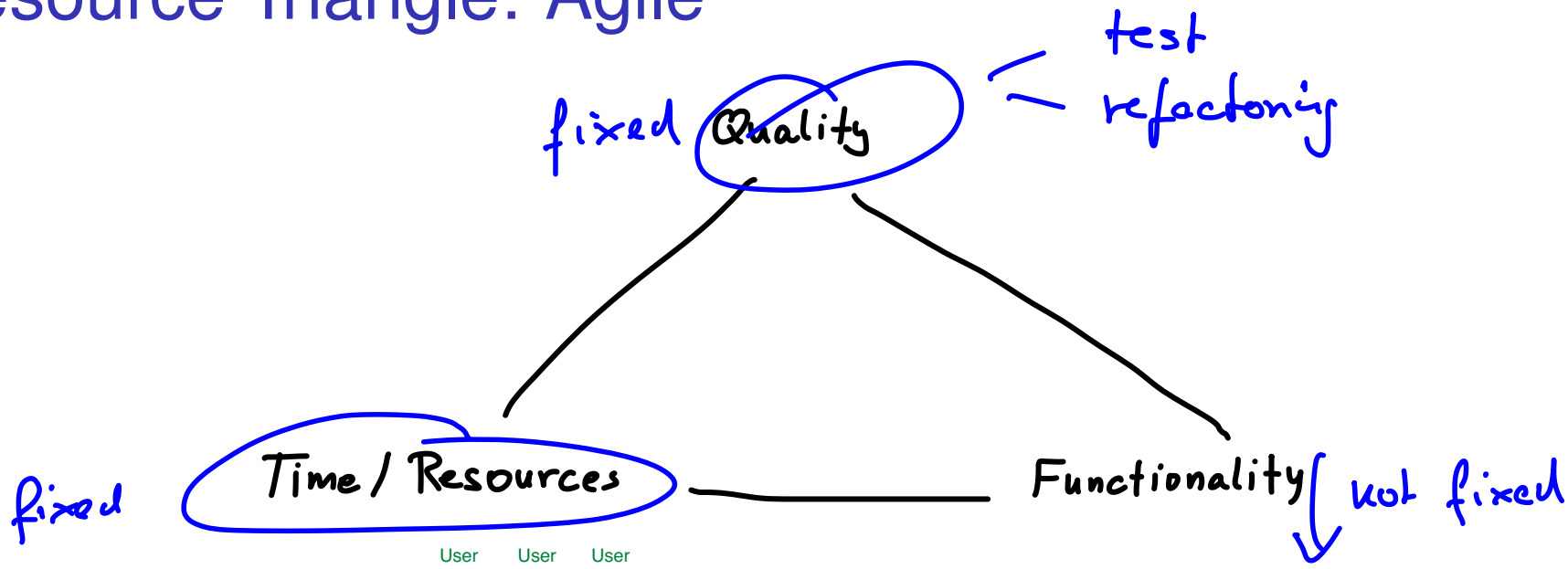


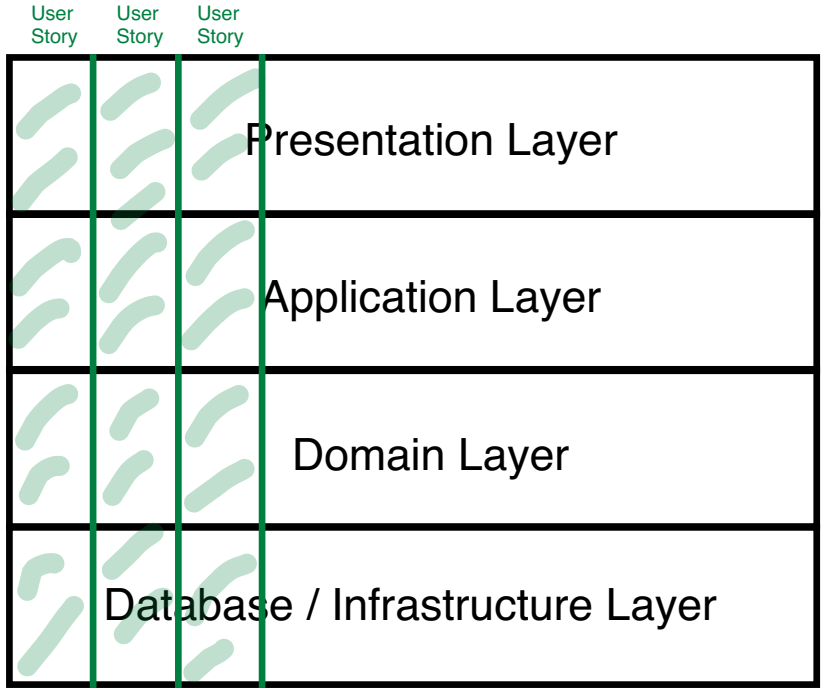
- Can only fix two of them at the same time

Resource Triangle: Waterfall

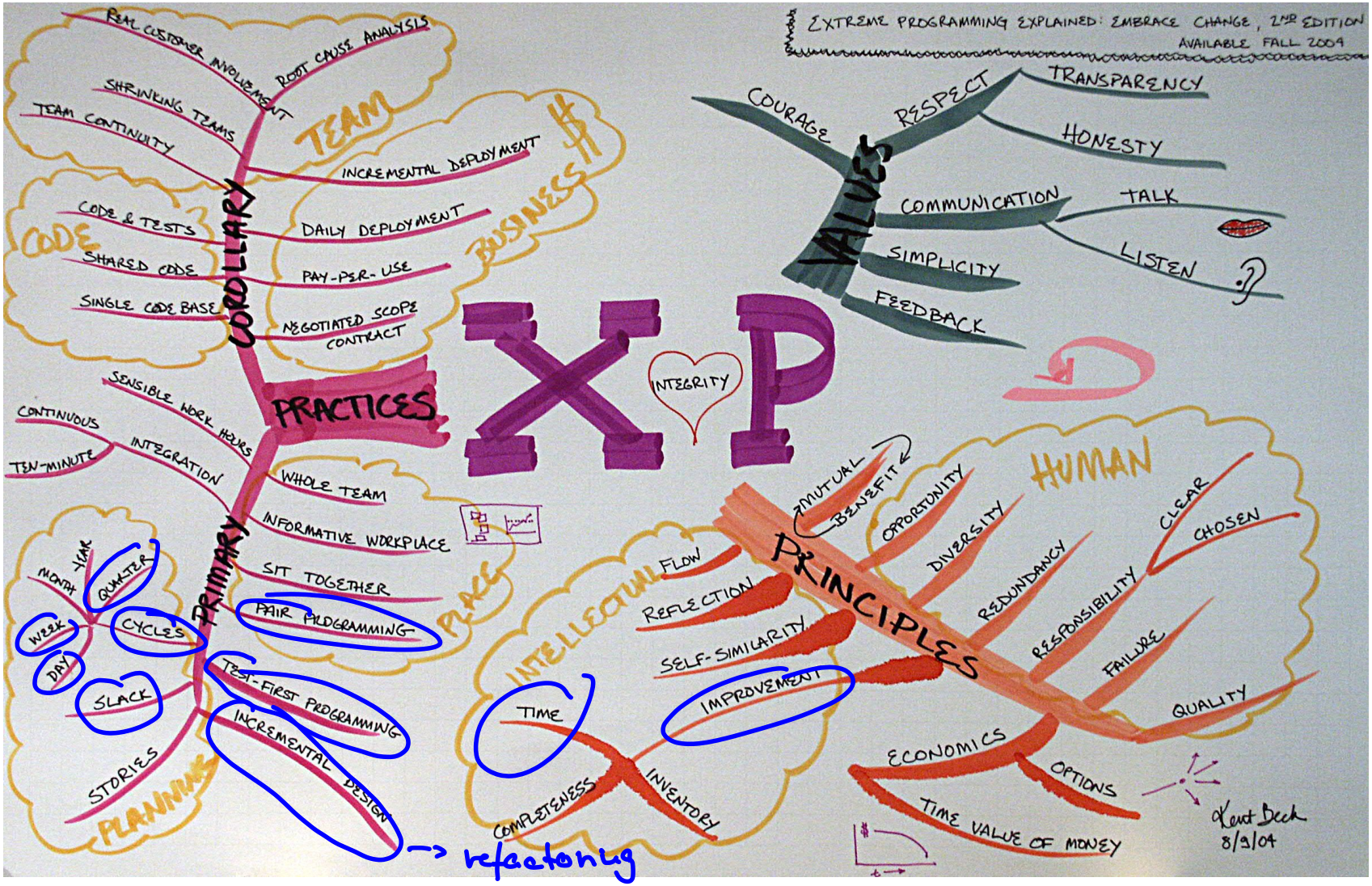


Resource Triangle: Agile



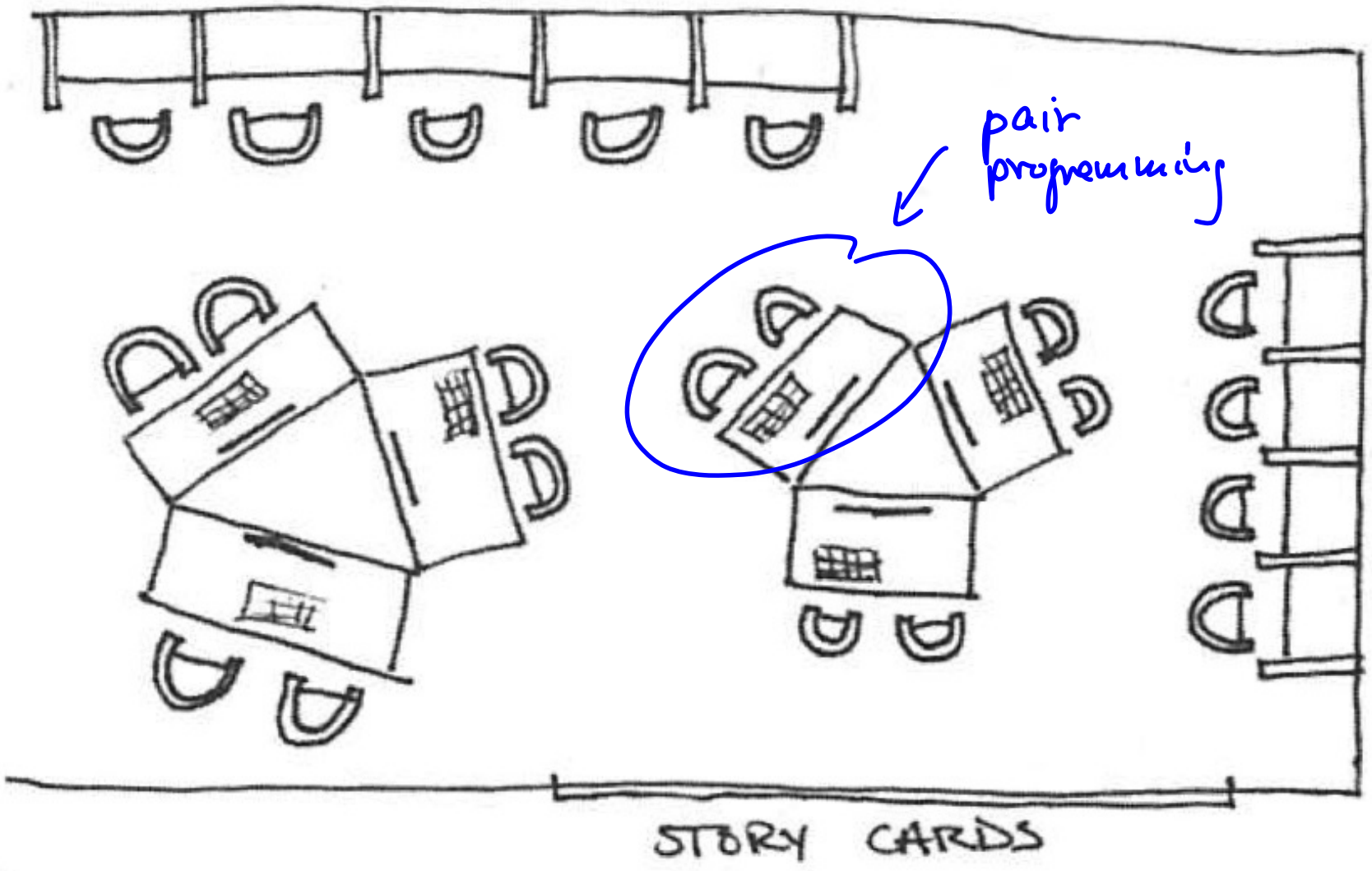


eXtreme Programming (XP)



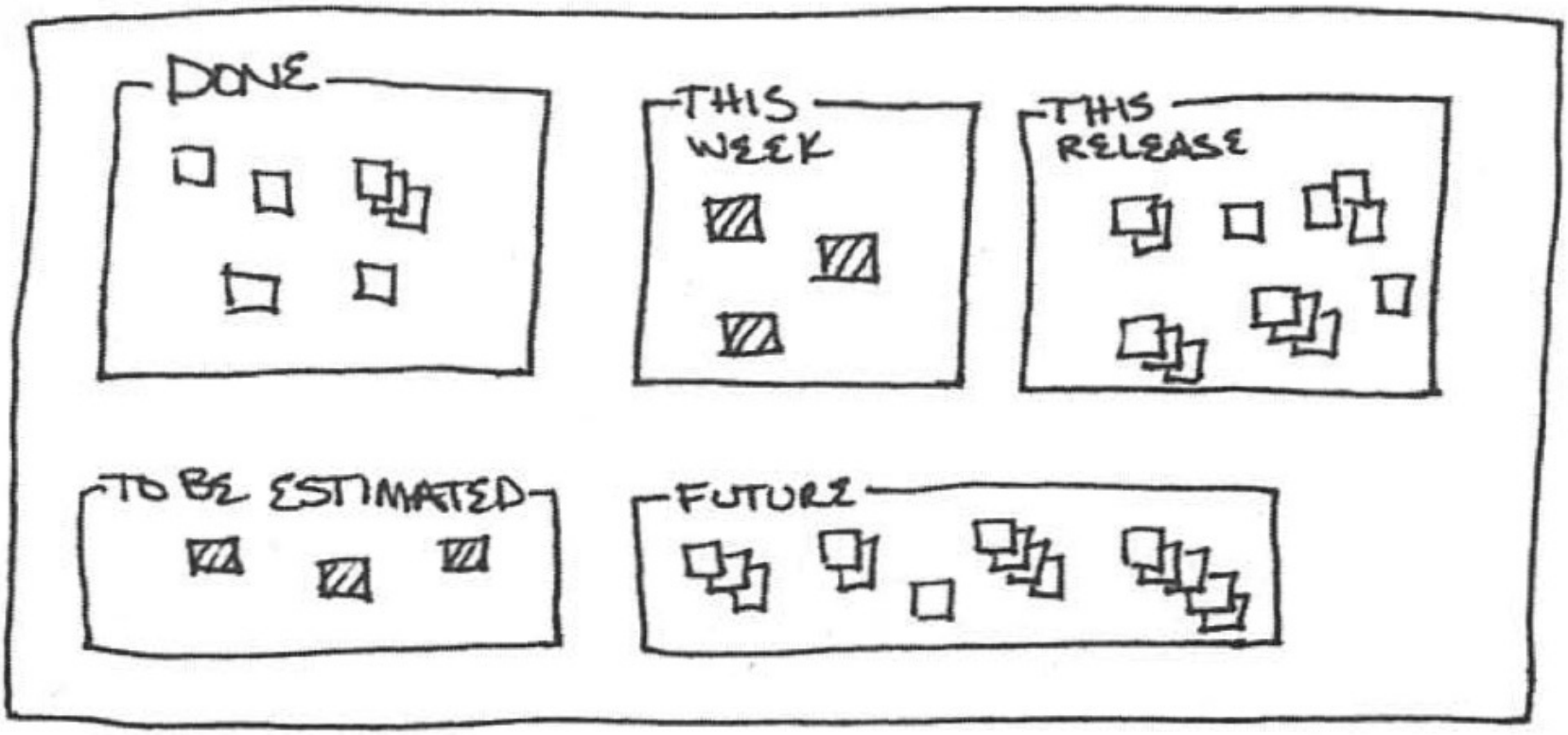
Kent Beck, Extreme Programming 2nd ed.

Sit-together



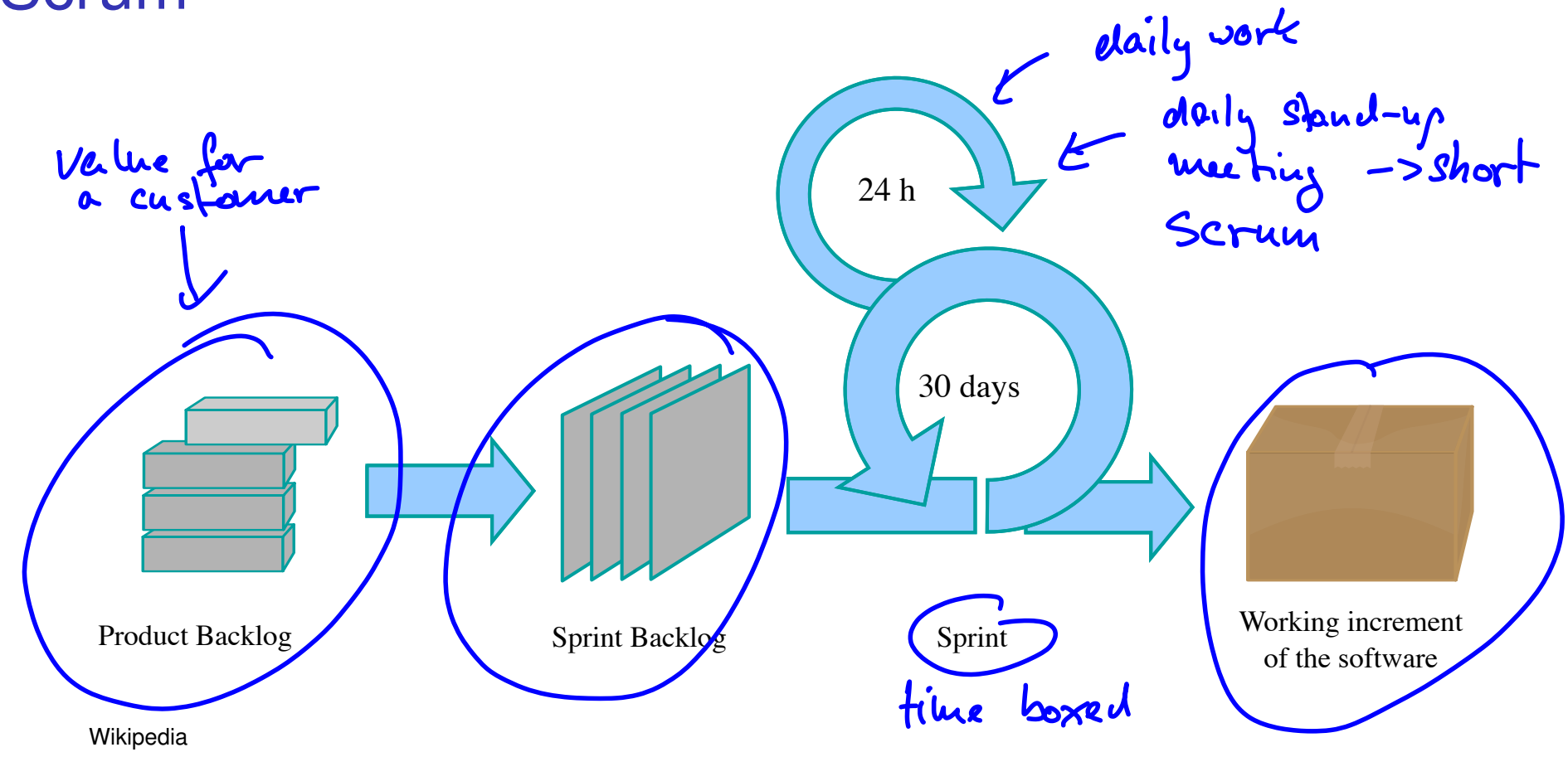
Kent Beck, Extreme Programming 2nd ed.

Visual wall



Kent Beck, Extreme Programming 2nd ed.

Scrum



- Robert Martin (Uncle Bob) about "The Land that Scrum Forgot"
  `http://www.youtube.com/watch?v=hG4LH6P8Syk`
  - → History about agile methods, the agile manifesto, and Scrum and its relationshop to XP

Lean Software Development

- Lean Production:
  - Reduce the amount of *waste* in the production process
  - Generate *flow*
- Waste: resources used which do not produce value for the customer
  - time needed to fix bugs
  - time to change the system because it does not fit the customers requirements
  - time waiting for approval
  - ...

Cycle time

Cycle time

Time it takes to go through the process one time

$$cycle\_time = \frac{number\_of\_features}{feature\_implemantion\_rate}$$

- Example: Waterfall
  - Batch size $=$ number_of_features in an iteration
  - Software: 250 features, feature_implementation_rate = 5 features/week
  - cycle_time = 250 f / (5 f/w) = 50 weeks
  - Overall time: 50 weeks

  $\rightarrow$ 1 cycle

Goal: Reducing the cycle time

- Reduce batch size: 1 feature in an iteration
- Software: 250 features, feature_implementation_rate = 5 features/week

$$cycle\_time = \frac{number\_of\_features}{feature\_implemantion\_rate}$$

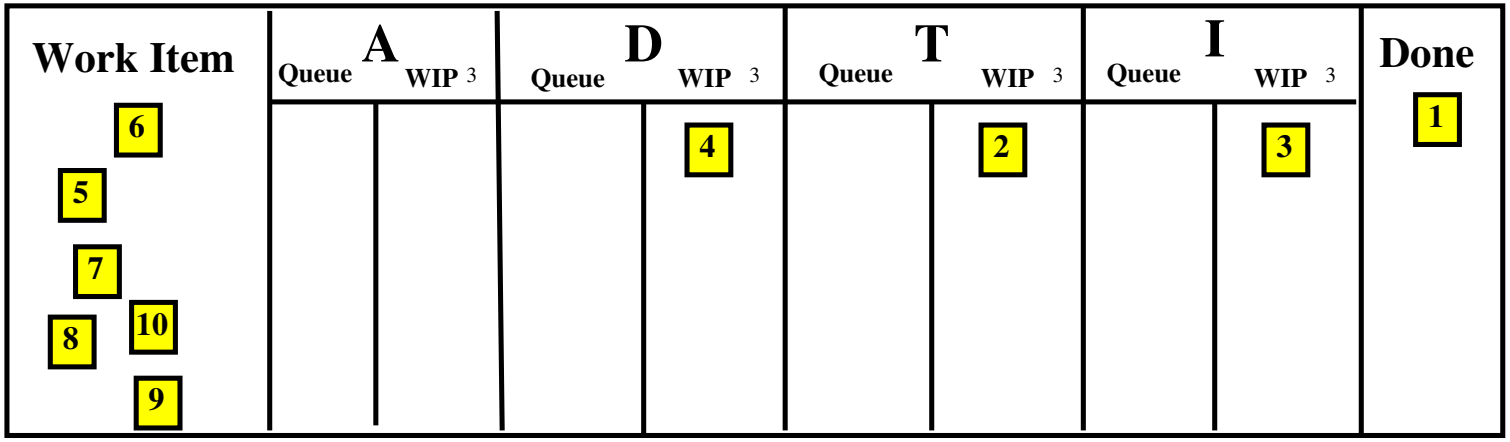
- Agile: cycle_time = 1 f / (5 f/w) = 1/5 week = 1 day = 8 h

$\rightarrow$ 250 cycles

- Advantages
  - Process adapts to changes in requirements
  - Process improvements and fine tuning

Generating flow using Pull and Kanban

WIP = Work in Progress Limit



Flow through Pull with Kanban

- Process controlling: local rules
- Load balancing: Kanban cards and *Work in Progress (WIP) limits*
- Integration in other processes

Figure from David Anderson `www.agilemanagement.net`

Online Kanban Tool: Trello

- `www.trello.com`: Electronic Kanban board useful for your project
- Example Kanban board `https://trello.com/b/4wddd1zf/kanban-workflow`

Contents

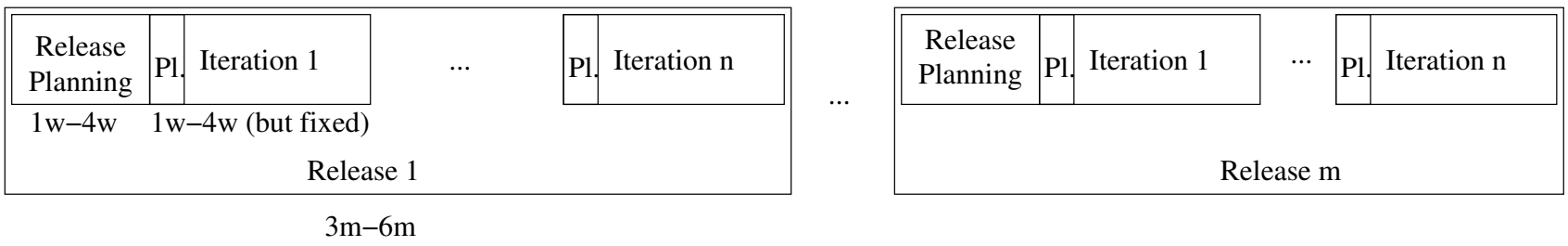
Project Planning

- Project plan
  - Defines:
    - How work is done
  - Estimate
    - Duration of work
    - Needed resources
    - → Price
- Project planning
  - Proposal stage
    - → Price
    - → Time to finish
  - Project start-up
    - → Staffing, . . .
  - During the project
    - Progress (tracking)
    - Adapt to changes

Planning Agile Projects

- *fixed* general structure

→ quarterly cycle / weekly cycle practices in XP / sprints in Scrum



- *time boxing*
  - fixed: release dates and iterations
  - adjustable: scope
- Planning: Which user story in which iteration / release

Planning game

- Customer defines:
  - user stories
  - priorities
- Developer define:
  - costs, risks
  - suggest user stories
- Customer decides: is the user story worth its costs?
  - $\rightarrow$ split a user story
  - $\rightarrow$ change a user story

Scrum/XP: User story estimation (based on ideal time)

- Estimation
  - Estimate *ideal_time* (e.g. person days / week) to finish a user story
  - real_time = ideal_time * load_factor (e.g. load_factor = 2)
  - Add user stories to an iteration based on real_time and priority
- Monitoring
  - New *load factor*: total_iteration_time / user_story_time finished

→ What can be done in the next iteration

- Yesterdays weather
  - only take load_factor from the last iteration for planning the next iteration
- Important: If in trouble focus on *few* stories and finish them

  → *Don't let deadlines slip* (time boxing)

Scrum/XP: User story estimation (based on points)

- Estimation
  - Estimate user stories *relative* to other user stories: *story_points*
  - *velocity*: number of story points that can be done in an iteration (initial value is a guess or comes from previous processes)
  - In an iteration: Select up to velocity amount of user stories
- Monitoring
  - *new_velocity*: story points of finished user stories per iteration

$\rightarrow$ What can be done in the next iteration

- user stories with story points up to new_velocity

Lean / Kanban: User story estimation

- No ”iterations”: user stories come in and flow through the system

→ Only a rough estimation of the size of the user stories
  - try to level the size of the user stories
  - Divide larger into smaller ones
- Measure process parameters, e.g., average cycle time
  - E.g. ”After committing to a user story, it takes in average a week to have the user story finished”
- User average_cycle_time and WIP (Work In Progress) Limit to determine the capacity of the process and thus throughput

Example of a Kanban board for the exam project

- `https://trello.com/b/iO29C07w/02161-example`

Next week

- Design by contract
- Basic principles of good design